5
Conclusion

Recent researches (HOCHSTEIN and LINDVALL, 2005; MACIA et al, 2012) have suggested that architectural erosion and drift processes are often intertwined. However, techniques for preventing architectural degradation tend to focus on solely supporting the detection of either erosion or drift symptoms. Moreover, there is not much empirical knowledge on how developers would benefit from unified support to detect both kinds of degradation symptoms. Given typical time constraints, developers are only encouraged to specify and maintain hybrid rules for drift and erosion if there is evidence about: (i) their effectiveness to reveal inter-related drift and erosion symptoms in software projects, and (ii) their potential reuse across several projects. The next subsection describes how the contributions of this dissertation overcome the aforementioned problems.

5.1.
Dissertation contributions

In this dissertation, we proposed **TamDera**, a language that supports hybrid specifications of anti-drift and anti-erosion rules. The language allows the detection of both categories of architectural degradation symptoms. This facility is not supported by existing techniques (Chapter 2). Different from them, **TamDera** does not require that developers learn two different languages to describe each set of anti-drift and anti-erosion rules.

Another contribution of **TamDera** is the explicit support for reusing both anti-drift and anti-erosion rules. **TamDera** provides features that enable the definition of anti-degradation rules that are not strictly defined for a particular project (Section 3.1.4). These features promote the compositional and hierarchical reuse of such rules. The latter structures groups of anti-degradation rules in hierarchy trees where each node can reuse or extend anti-degradation rules from its parent. Other techniques do not support the reuse and extension of architectural rules in multiple contexts. This is particularly interesting given the fact that anti-

degradation rules usually have their thresholds and dependency constraints adjusted in each project (MARINESCU, 2004; MOHA et al, 2010).

A third contribution of this dissertation is the tool supporting the automatic enforcement of **TamDera** rule specifications. The language has been supported by a prototype that detects degradation symptoms in Java systems (Section 3.2). The tool design allows the extension of the set of dependency types and the metrics available respectively for anti-drift and anti-erosion rules (Section 3.2.2). This may foster the tool integration with other forms of anti-drift rules as the **TamDera** tool can be extended to process measures collected from other measurement tools.

In addition, the tool verifies some cases that reveal inconsistent sets of rules (Section 3.2.3). Architects can accidently define inconsistent anti-erosion rules. These rules involve contradictory constraints (Section 3.2.3) or even the establishment of cyclic dependencies (TERRA and VALENTE, 2009). Thus, anti-degradation tools may wrongly indicate degradation symptoms which are indeed not related to any degradation process. In this fashion, architects spend time and resources manually checking the implemented architecture and the architecture specifications. As far as we know, none of other anti-erosion techniques supports this facility (Section 2.2).

Finally, there is not much knowledge in the literature about the benefits of supporting joint detection of drift and erosion problems. Our exploratory study represents a first effort to address this gap. The dissertation presents in detail a systematic study, which is also tailored for evaluating the usefulness of **TamDera** (Chapter 4). We detected degradation symptoms in 21 versions of 5 different projects. Our evaluation showed that **TamDera** could help architects and developers to save time and resources. We found that 72% of the anti-degradation rules were reused. Many of them were naturally of hybrid nature, i.e., both drift and erosion rules associated with a single architectural concept (Section 4.4.2). Typical examples were hybrid rules associated with concepts derived from architectural styles or design patterns (Section 4.4.2).

The analysis also revealed that such reused rules were responsible for identifying on average 75% of all degradation symptoms in the target projects (Section 4.4.1). More interestingly, many reused rules were effective to detect the same kind of degradation symptom across multiple projects (Section 4.4.2). There were a broad range of scenarios confirming that individual techniques for drift or

erosion (Section 4.4.1) would not be sufficient or efficient to support degradation prevention. For instance, in cases were drift and erosion symptoms were affecting the same module in the code, developers detected and removed one form of degradation symptom in a certain version, but the other remained (Section 4.4.1) and, thereby, the system continuously degraded through the residual degradation process.

In summary, the main contributions of this dissertation are: (i) the design of the **TamDera** language to support the specification of hybrid rules for preventing architectural erosion and drift; (ii) the support in **TamDera** to support reuse of such rules; (iii) a prototype tool that implements the language and supports automatic rule enforcement, and; (iv) an exploratory study about the synchronized manifestation of erosion and drift symptoms as well as the **TamDera** adequacy to better support their detection and reuse of anti-degradation rules.

5.2. Future work

As part of our future plans, we intend to investigate to what extent **TamDera** supports developers on the conception of anti-degradation rule specifications that are resilient to code changes throughout the system evolution. In particular, the goal is to have insights about how often definitions of architectural concepts need to change in order to accommodate modifications or additions of code elements as well as design refactorings. It might be, for instance, that concept mappings need to be often updated to consider such code and design changes. It might be that thresholds need to be often revisited as well. In this fashion, we would have more evidence about the **TamDera** adequacy to define and reuse hybrid rules and also to what extent these rules are easily maintained or not.

We also plan to build a library of hybrid rules for a wide range of architectural styles and design patterns which are relevant to architecture such as Façade (GAMMA et al, 1995). The goal is to foster extensive reuse of anti-degradation rules which are related to patterns. More specifically, we plan to describe hybrid rules derived from descriptions of architectural styles (BUSCHMANN et al, 2007; CLEMENTS et al, 20120) and design patterns

(GAMMA et al, 1995). Throughout our exploratory study, we already specified reusable rules for a few design and architectural patterns, such as the design patterns Mediator (Section 3.1.5) and Chain of Responsibility (Section 4.4.2), as well as the architectural pattern Model-View-Controller (Section 2.3). Their specification provided us with insights to elaborate the reuse mechanisms of **TamDera** (Section 3.1.4).

TamDera's abstractions rely on: pseudo-natural statements for anti-erosion rules, mathematical expressions for anti-drift rules, and reuse mechanisms. These abstractions and mechanisms seem quite intuitive and easy to use and understand. However, we did not evaluate the **TamDera** language in terms of its usability. Evaluations with this goal are also included in our agenda.

Even though **TamDera** supports the reuse of anti-degradation rules in multiple contexts, there is still a considerable effort to specify the rules. More specifically, architects need to select which concepts and rules from a source of reusable rules (e.g., library) fit the architecture properties of a particular system. If they do not exactly fit the purpose, they need to be specialized or overridden, which require some additional effort. Therefore, we plan to design and execute empirical studies in order to perform trade-off analysis contrasting benefits and effort in different project contexts. In particular, we also plan to study these issues as well as the expressive power of **TamDera's** mechanisms to support variability in program family architectures (FIGUEIREDO et al, 2008).

Finally, we also plan to improve the **TamDera** language in three directions. First, we aim to provide more flexibility to define the concept mapping. The idea is to enable architects to elaborate queries whose returned code elements comprise the architectural concept. These queries may use similar constructions with this purpose supported by Semmle Code (Section 2.5.3). Second, we intend to improve our concept mapping mechanism even further by allowing concept definitions to use the same mapping of another concept. This is particularly useful when architects assign to the same module several constraints imposed by different architectural concepts (e.g., a module that realizes several roles of different design patterns). Third, we plan to provide features for supporting architects to constrain the concept mappings and hierarchy relationship among architecture concepts. For instance, users could specify that the sets of code elements pertaining to two concepts are disjoint. This facility would be useful, for

instance, when these concepts refer to components comprising two alternative features (FIGUEIREDO et al, 2008) of a software product line architecture.